Logique sur les graphes et protocoles de diffusion

Damien Regnault

September 18, 2003

1 Introduction

Ce stage s'est déroulé au Loria de Nancy sous la direction de Johanne Cohen (équipe Algorille) et de Olivier Bournez (équipe Prothéo).

Dans sa thèse, Johanne Cohen a découvert un algorithme pour résoudre en temps polynomial un problème de diffusion dans les arbres(problème NP-complet dans le cas de diffusion dans les graphes (voir la thèse de Johanne Cohen)). Le but de ce stage est d'apporter une généralisation de cet algorithme des arbres vers les graphes de tree-width borné.

Nous avons essayé de généraliser le problème dans le cas des graphes à treewidth borné et des pathwidth pour pouvoir utiliser le théorème de Courcelles (voir Parametrized Complexity).

Ce théorème donne directement un algorithme linéaire pour résoudre des problèmes de décision sur les treewidth lorsque nous pouvons caractériser ce problème par une formule d'une certaine logique.

Nous avons donc chercher à comprendre la façon dont cet algorithme était généré.

Puis nous avons essayé de voir les résultats que donnait ce théorème sur notre problème.

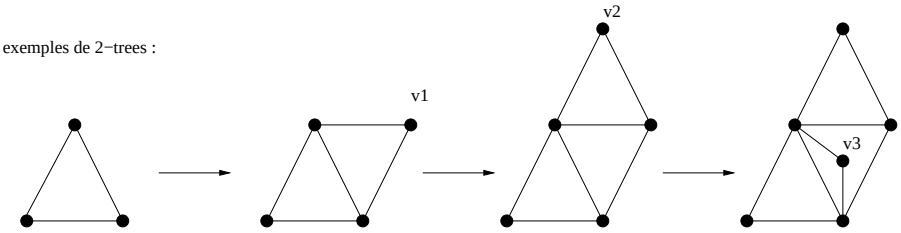
Enfin, nous avons essayé de concevoir un algorithme.

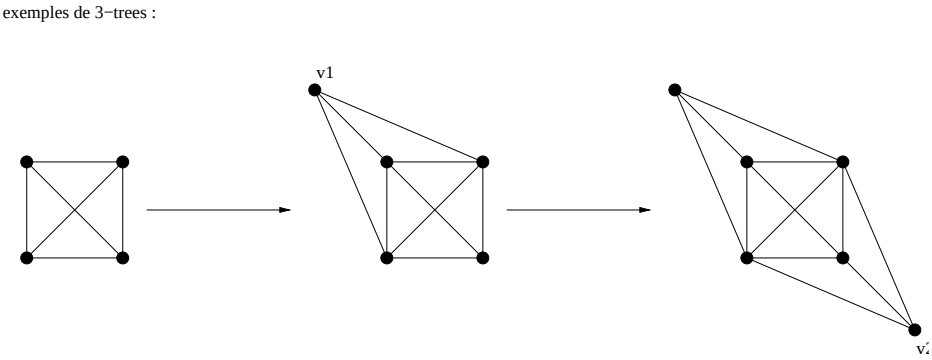
2 Les pathwidth et les treewidth

Nous allons présenter les graphes sur lesquels nous avons travaillé. Les graphes à treewidth borné sont des graphes qui peuvent être construits de façon récursive, ce qui permet d'aborder des problèmes sur ces graphes de manières différentes. Tout d'abord nous allons introduire la notion de la treewidth d'un graphe et des éléments nécessaires à cette définition.

Définition 1 *Un graphe est un k-tree s'il peut se construire de la façon suivante:*

On part de la clique de taille k+1, on rajoute un sommet, on sélectionne k autres sommets qui forment une clique et on complète le graphe de façon que les k sommets sélectionnés et le nouveau forment une clique de taille k+1, et ainsi de suite(voir Figure1).





L'aspect récursif de la construction est bien présent ici. La treewidth d'un arbre faisant appel à la notion de sous-arbre nous allons faire un rappel de cette définition avant de donner la définition.

Rappel 1 *Si G=(V,E) est un graphe alors G'=(V',E') est un sous-graphe de G si et seulement si V' est inclus dans V et si E'est inclus dans E et que toutes les arêtes de E' relient des sommets de V'.*

Maintenant, nous pouvons définir la notion de treewidth.

Définition 2 *La treewidth d'un graphe G est le plus petit k tel que G soit un sous-graphe d'un k-tree.*

Les familles de graphes à treewidth borné ne sont pas marginales ainsi de nombreuses familles de graphes sont à treewidth borné, par exemples:

-les arbres sont de treewidth 1.

-les graphes planaires de rayon k sont de treewidth 3k.

-les bandwidth et les cutwidth de taille k sont des treewidth de taille k.

...

Maintenant nous allons voir comment un problème de décision sur ces familles de graphes peut être résolu sur ces arbres en temps linéaire par le théorème de Courcelles.

3 Théorème de Courcelles

Le théorème de Courcelles prend en entrée une formule de la logique monadique du second ordre et donne un algorithme pour savoir si un graphe vérifie cette propriété. En fait, à partir de la formule, le théorème va concevoir un automate d'arbre. Les graphes de treewidth borné peuvent se construire par l'application successive d'un certain nombre d'opérateurs. Cette construction peut se représenter sous la forme d'un arbre et peut être calculée en temps linéaire à partir du graphe initial. C'est donc sur cet arbre que va être appliqué l'automate d'arbre.

Premièrement, nous allons faire un rappel sur les automates et leurs propriétés. Puis nous expliquerons le fonctionnement des automates d'arbres et leur rapport avec les automates. Ensuite nous présenterons la façon dont les graphes sont traités pour être transformés en arbres et ainsi tirer profit de leur construction récursive. Enfin, nous présenterons le théorème de Courcelles ainsi que les idées de sa preuve.

3.1 Automates

On utilise ici la définition classique des automates, c'est à dire :

Rappel 2 *un quintuplet(K,Σ,δ,S,F) où:*
-K est l'ensemble des états
-Σ est un alphabet.
-δ est un fonction de $K * \Sigma \to K$ appelée fonction de transition.
-S est l'ensemble des états initiaux.
-F est l'ensemble des états finaux.

Le théorème de Courcelles utilise un dérivé du théorème de Myhill Nérode qui est basé sur les propriétés de la relation suivante:

Définition 3 *Soit Σ un alphabet et L un langage sur Σ et a et $b \in \Sigma^\star$.*
$\sim$ est une relation binaire tel que $a \sim b$ si et seulement si $\forall\ c \in \Sigma^\star$ alors $ac \in L$ si et seulement si $bc \in L$.

Le théorème suivant appelé théorème de Myhill-Nérode raisonne sur les classes d'équivalence de cette relation:

Théorème 1 *Un langage est reconnaissable si et seulement si le nombre de classes d'équivalence pour la relation $\sim$ est fini.*

Bien que nous ne parlerons plus d'automates, nous utiliserons des équivalents de cette relation et de ce théorème dans d'autres modèles.

3.2 Automates d'arbres (tree automata)

Nous allons maintenant présenter les automates sur lesquels nous allons travailler.

Définition 4 *Un automate d'arbres sur Σ est un quintuplet (K,Σ,δ,S,F) et un entier f où K,S,F sont définis comme dans les automates classiques. Mais δ est une ici une fonction de $(\underbrace{K * K * .. * K}_{} * \Sigma) \rightarrow K$. Cet automate prend en entrée un arbre (dont le nombre de fils par noeud est borné par f) auquel on a associé à chaque noeud un éléments de Σ. Il le parcourt des feuilles à la racine, à chaque feuille, il fait correspondre l'état initial et à chaque noeud, il fait correspondre le résultat de la fonction γ appliqué aux états des fils et à la lettre associée au noeud.*

Dans l'exemple ci-contre, l'évaluation de l'arbre va donner:
$\delta(1, \delta(0, \delta(1, \delta(1, \delta(1)), \delta(0)), \delta(1, \delta(1), \delta(1))), \delta(0, \delta(1, \delta(1)), \delta(0, \delta(0), \delta(1))))$
$\delta(1)$ et $\delta(0)$ signifie en fait $\delta(1, q_0)$ et $\delta(0, q_0)$
ceci sera évalué en:
$\delta(1, \delta(0, \delta(1, \delta(1, q_1), q_0), \delta(1, q_1, q_1)), \delta(0, \delta(1, q_1), \delta(0, q_0, q_1)))$
puis en:
$\delta(1, \delta(0, \delta(1, q_1, q_0), q_1), \delta(0, q_1, q_0))$
puis en :
$\delta(1, \delta(0, q_0, q_1), q_0)$
puis en :
$\delta(1, q_0, q_0)$
et enfin cet arbre sera évalué en q_1 par l'automate d'arbre.

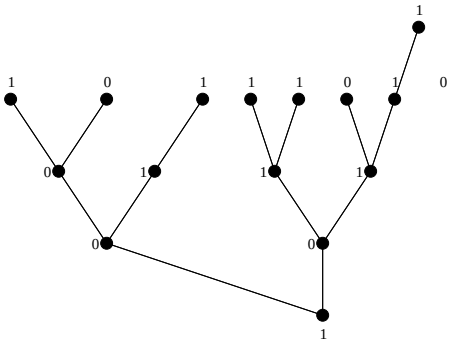
Maintenant, nous allons nous intéresser aux analogies avec le théorème de Myhill-Nérode dans ce nouveau modèle. Pour cela nous allons commencer par redéfinir la relation $\sim$ dans le cas des automates d'arbres.

Définition 5 *Soit F une famille de graphes définie sur Σ. Soient a et b deux arbres sur Σ ayant chacun une feuille notée x avec $x \notin \Sigma$.$a \sim b$ si et seulement si pour tout arbre c défini sur Σ, l'arbre a qui aurait la feuille x remplacée par c appartient à F si et seulement si l'arbre b qui aurait la feuille x remplacée par c appartient à F.*

Avec cette relation le théorème de Nyhill-Nérode est toujours valable.

Maintenant, nous allons expliquer comment trouver la représentation d'un graphe à treewidth borné sous forme d'arbre. Pour cela nous allons avoir besoin de définir des opérateurs, ces opérateurs sont définis sur des t-boundaried graph. Nous allons donc auparavant présenter ces graphes pour pouvoir introduire ces opérateurs.

Définition 6 *Un t-boundaried graph est un graphe qui a t sommets distingués numérotés de 1 à t.*



States	Symbols	δ (states,symbols)
q0	0	q0
q0	1	q1
q1	0	q1
q1	1	q1
q0,q0	0	q0
q0,q0	1	q1
q0,q1	0	q0
q0,q1	1	q1
q1,q0	0	q0
q1,q0	1	q0
q1,q1	0	q0
q1,q1	1	q1

Définition 7 *On définit 6 opérateurs sur les t+1-boundaried graph par:*

-$\emptyset$: crée t+1 sommets numérotés de 1 à t+1.

-γ: opérateur unaire qui fait la permutation: $j \to j+1$, $t+1 \to 1$ sur la numérotation des sommets du graphe.

-i: opérateur unaire qui intervertit les sommets 1 et 2.

-e: opérateur unaire qui rajoute une arête entre les sommets 1 et 2.

-u: opérateur unaire qui ajoute un sommet numéroté 1 et qui décale la numérotation des autres sommets (le sommet t+1 devient non numéroté).

-$\oplus$: opérateur binaire qui fait la concaténation des deux graphes en les liant par les sommets (le sommet t+1 devient non numéroté).

Nous pouvons maintenant caractériser les arbres qui vont nous permettre de représenter les graphes à treewidth borné. Ces arbres sont appelés graphes couvrants(Parsing graphs).

Théorème 2 *Tout arbre de tree-width t peut s'écrire comme une composée des opérateurs de la définition 7. Un arbre syntaxique correspondant est appelé graphe couvrant.*

Lorsque t est fixé, il existe un algorithme linéaire qui retourne pour tout graphe de tree-width t un graphe couvrant.

Maintenant avant de pouvoir donner le théorème de Courcelles, il faut définir la logique sur laquelle nous allons travailler pour caractériser une famille de graphes. Cette logique est la logique monadique du second ordre appliquée à la théorie des graphes:

Définition 8 *C'est la logique constituée:*

-des connecteurs logiques $\wedge, \vee, \neg$.

-des variables dénotant des sommets, des arêtes, des ensembles de sommets et des ensembles d'arêtes.

-des quantificateurs $\forall, \exists$.

-des 5 relations binaires suivantes:

1)$d \in S$: d est une variable représentant une arête et D une variable représentant un ensemble d'arêtes.

2)$u \in U$: u est une variable représentant un sommet et U est une variable représentant un ensemble de sommets.

3)inc(d,u): d et u sont des variables représentant des arêtes et l'interprétation de cette relation est que d et u sont adjacents.

4)adj(u,v): u et v sont des variables représentant des sommets et l'interprétation de cette relation est que u et v sont adjacents.

5)l'égalité pour les arêtes, les sommets, les ensembles d'arêtes et les ensembles de sommets.

Maintenant que tous les éléments sont en place nous pouvons passer au théorème de Courcelles:

Théorème 3 *Tout problème de décision P sur les graphes, tel qu'il existe une formule ϕ de la logique monadique du second ordre tel que P soit l'ensemble des graphes satisfaisant ϕ, est résoluble en temps linéaire si l'on se restreint aux graphes de tree-width borné par t pour tout t.*

Ceci n'est qu'un squelette de la vraie preuve du théorème de Courcelles:

Preuve:

La démonstration est basée sur le fait que les graphes de treewidth borné peuvent être construits de manière récursive comme dans la définition des k-tree (Voir Partie1) à la différence que lors du rajout du nouveau sommet, les arêtes rajoutées ne forment pas forcément une clique.

Une série de tests (tests set) va être générée à partir de la formule logique, cette série correspond à toutes les façons dont peut influencer la nouvelle sous-clique que l'on ajoute. Ainsi si la formule contient ∀u ou ∃u (u arête ou sommet), la série de tests contiendra des tests pour les emplacements possibles de cette arête (ou sommet) dans une clique de taille t+1. Pour un ensemble de sommets ou d'arêtes, on associe une couleur sur les sommets(ou arêtes) et de même on rajoute ainsi une nouvelle combinaison de tests. S'il y a plusieurs quantificateurs, il faut traiter toutes les combinaisons possibles entre les différents tests.

On obtient ainsi $2^{\text{le nombre de tests générés}}$ classes d'équivalences: en effet un graphe donné ne validera qu'une partie des tests, ces tests seront caractéristiques de sa classe d'équivalence.

Comme le nombre de classes d'équivalences est fini, le théorème de Nyhill-Nérode permet de construire un automate résolvant ce problème en temps linéaire. Néanmoins, cet automate s'applique sur les graphes couvrants et non sur les graphes initiaux, mais comme le passage de l'un à l'autre se fait en temps linéaire, l'algorithme reste linéaire.

C'est lors de la génération de l'automate que l'on se sert de la formule pour créer les transitions et calculer les états entrants et les états finaux.

□

Nous allons donner des exemples pour mieux comprendre la preuve de Courcelles. Ce sont les tests pour les formules élémentaires.

Dans le dernier cas lors du test de inc(e,v), tous les tests n'ont pas été dessinés.

Exemple 1 (Application) *Décider si un graphe est hamiltonien est résoluble en temps linéaire lorsqu'on se restreint aux graphes de tree-width borné.*

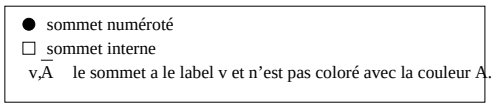
En effet, voici la formule logique qui permet de savoir si un graphe est hamiltonien:

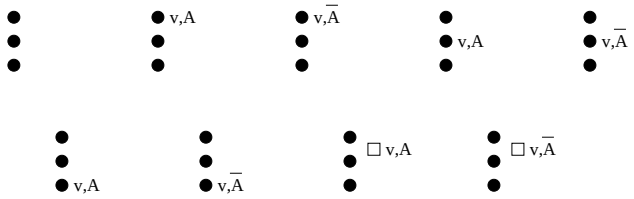
$\exists R, \exists B, \forall u, \forall v, [part(R,B) \wedge deg(u,R) = 2 \wedge span(u,v,R)]$ ou $span, deg$ et $part$ sont décrits de cette façon:

$part(R,B) \equiv \forall e[(e \in R \vee e \in B) \wedge \neg(e \in R \wedge e \in B)]$.

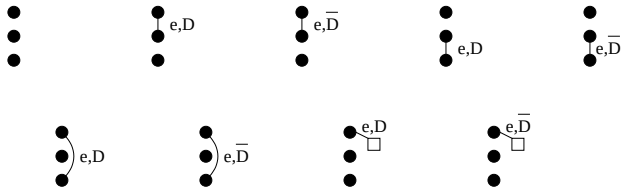
$deg(u,R) = 2 \equiv [\exists e_1, e_2(e_1 \neq e_2 \wedge inc(e_1,u) \wedge inc(e_2,u) \wedge e_1 \in R \wedge e2 \in R)] \wedge \neg[\exists e_1, e_2, e_3(e_1 \neq e_2 \neq e_3 \wedge (inc(e_i,u) \wedge e_i \in R \text{ pour } i \in 1,2,3))]$.

$span(u,v,R) \equiv \forall V, W[(part(V,W) \wedge u \in V \wedge v \in W) \rightarrow (\exists e, x, y(inc(e,x) \wedge inc(e,y) \wedge x \in V \wedge y \in W \wedge e \in R))]$.

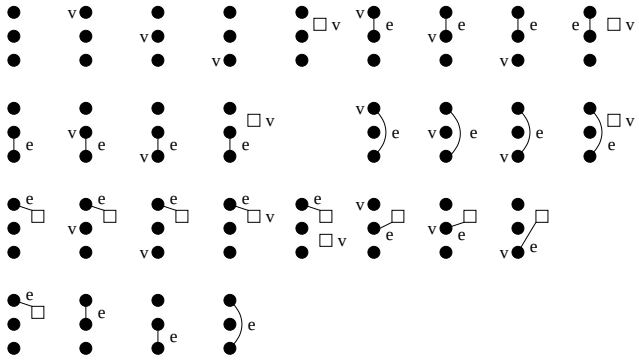




série de tests pour $v \notin A$, où v est une variable représentant un sommet et A une variable représentant un ensemble de sommet



série de tests pour $e \notin D$, où e est une variable représentant une arête et D une variable représentant un ensemble d'arete.



série de tests pour inc(e,v), où e est une varible représentant une arête et v une variable représentant un sommet.

Ces formules servent à prouver que résoudre le problème du chemin hamiltonien sur un graphe à treewidth borné est faisable en temps linéaire.

4 Problème de diffusion

4.1 Présentation du problème

Nous allons maintenant présenter le problème particulier que nous avons voulu résoudre. Pour cela, il est nécessaire de savoir ce qu'est un problème de diffusion. Les différents modèles envisageables étant nombreux, nous allons dire celui sur lequel nous avons travaillé et enfin présenter l'algorithme qui a déjà été trouvé par Johanne Cohen. Nous nous plaçons dans le cadre du modèle commuté Δ-port arêtes disjointes qui modélise des protocoles comme ATM

Nous allons donc commencer par expliquer ce qu'est une diffusion sur un graphe dans le cadre du modèle commuté:

Définition 9 *Soit G=(V,E) un graphe. Soit $u \in V$ appelé source, c'est le seul sommet informé au début de la diffusion. Une étape est un ensemble de chemin dans E. Une diffusion dans G est un ensemble de k étapes appelé protocole respectant les règles:*

-tous les chemins dans une même étape vérifient le fait que l'une de leurs extrémités est un sommet informé et l'autre non. A la fin d'une étape toutes les extrémités des chemins non informées deviennent informées.

-tous les sommets de V doivent être informés à la fin de la dernière étape.

-les étapes doivent respecter un certain nombre de règles selon le modèle choisi.

Un protocole est optimal lorsqu'il est valide et que tout protocole valide a au moins autant d'étapes que ce protocole.

Le modèle étudié ici est la diffusion Δ-port arêtes-disjointes.

Voici la définition d'arêtes-disjointes:

Définition 10 *Une étape est arêtes-disjointes si et seulement si tous les chemins n'ont aucune arête en commun avec un autre chemin lors de la même étape.*

Voici la définition de Δ-port:

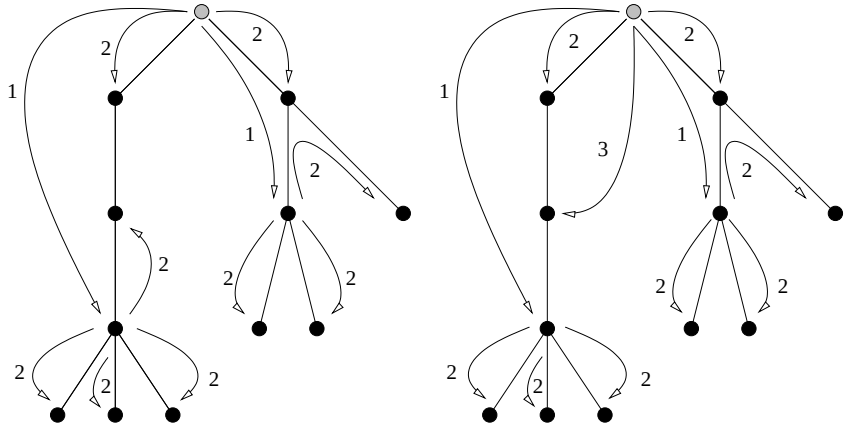
Définition 11 *Une diffusion est Δ-port lorsque un sommet peut être une extrémité de plusieurs chemins différents lors d'une même étape (s'oppose à 1-port où un sommet ne peut être l'extrémité que d'un seul chemin lors d'une étape).*

Dans ce modèle, le problème de diffusion Δ-port est NP-complet.

Voici quelques exemples de diffusion Δ-port:

Dans ces exemples, le protocole de gauche fait la diffusion en 2 étapes tandis que le protocole de droite fait la diffusion en 3 étapes.

Dans ce graphe, il est impossible d'effectuer la diffusion en une seule étape, donc le protocole de gauche est optimal et celui de droite ne l'est pas.



exemples de diffusion $\triangle$ −port dans un arbre.

4.2 Algorithme dans le cas des arbres

Il existe un algorithme pour résoudre ce problème dans le cas des arbres. Il faut d'abord fixer des notations pour pouvoir raisonner sur les protocoles.

Algorithme 1 *Dans ce problème, la source est considérée comme la racine de l'arbre. Cet algorithme renvoie un protocole de diffusion optimal sur un arbre tel que tous les sommets soient informés le plus tard possible.*

Sur une arête, le protocole à une étape donnée peut soit faire passer une information de haut en bas (ce cas est codé par 1), soit faire passer aucune communication (ce cas est codé par 0), soit faire passer une communication de bas en haut(ce cas est codé par -1).La trace est un vecteur associé à une arête, de taille égale au nombre d'étapes du protocole et sa valeur en i est le codage de la communication qui passe dans l'arête à l'étape i.

Le protocole de diffusion est construit des feuilles en remontant à la racine. Les arêtes reliant les feuilles à leur père reçoivent la trace [0;0;...0;1]. De cette façon elles sont prévenues lors de la dernière étape. Pour un noeud, l'algorithme va créer la trace de l'arête reliant le père à son père en analysant la trace des arêtes reliant les fils au père. Pour cela, il va regarder les traces étape par étape:

A un appel à l'extérieur (1), il fait correspondre un envoi vers l'extérieur (-1) d'un autre de ses fils. Au final, on regarde les appels restants:

-Deux envois ou plus vers l'extérieur($\leq$-2): le père peut en utiliser un pour s'informer et en envoyer un vers l'extérieur. Une fois le père informé, les appels à l'extérieur peuvent être tous remplacés par des appels au père et le père peut toujours émettre un envoi vers l'extérieur.

-Un envoi vers l'extérieur (-1): Le fait que l'on peut informer le père à ce moment en consommant l'appel est mémorisé, mais pour le moment un signal

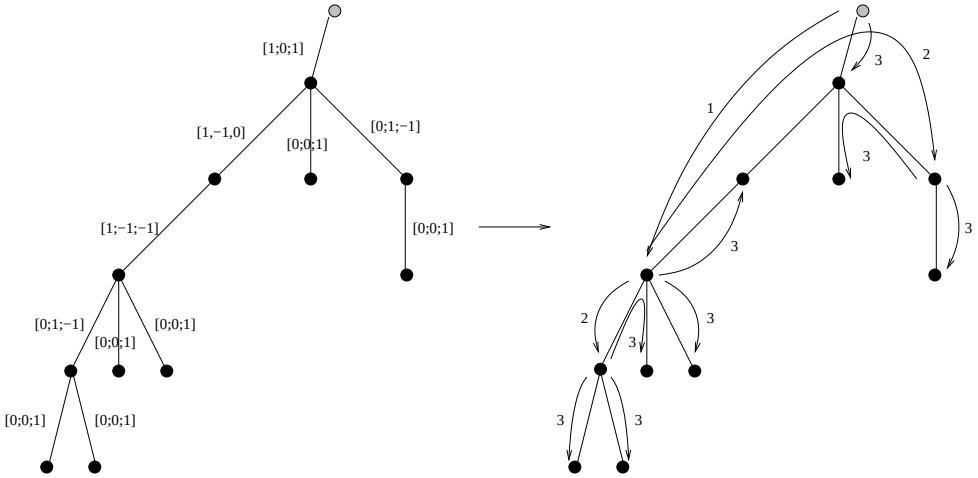
est envoyé vers l'extérieur et l'algorithme regarde s'il existe un meilleur moment pour avertir le père.

-Aucun appel(0): Le fait que l'on peut informer le père à ce moment en faisant un appel vers l'extérieur est mémorisé, mais pour le moment aucun appel n'est envoyé ou réceptionné et l'algorithme regarde s'il existe un meilleur moment pour avertir le père.

-Un appel vers l'extérieur(1):L'algorithme est obligé de faire un appel vers l'extérieur à ce moment.

-Deux appels vers l'extérieur($\geq$2):Le seul moyen de satisfaire cette demande est d'avoir informé le père auparavant, donc l'algorithme s'arrête et regarde le meilleur moment pour informer le père. S'il n'y en a pas alors il rajoute une étape au protocole et avertit le père lors de la première étape.

Voici un exemple de l'application de cet algorithme.



application de l'algorithme de diffusion $\triangle$−port dans un arbre

Les idées essentielles de cet algorithme sont que:

-on peut avoir un ordre total sur les traces(l'ordre lexicographique) tel que s' il existe un protocole valide optimal avec une trace donnée alors pour toute trace plus grande que cette trace, il existe un protocole optimal et valide qui a cette trace.

-Seules les traces reliant les fils au père sont nécessaires pour remonter dans l'arbre. Autrement dit, les traces représentent des classes d'équivalences et le nombre de traces possibles est bien fini d'où le lien avec le théorème de Courcelles.

4.3 Résultat de l'application du théorème de Courcelles

Le théorème de Courcelles nous a permis de résoudre en temps linéaire le problème suivant:

- **Données**: G=(V,E) un graphe, k $\in \Re$, m $\in \Re$, u $\in$ V
- **Question**: Existe-t-il une diffusion Δ-port arêtes-disjointes dans G avec comme source u en moins de k étapes et avec moins de m chemins simultanés par étape?

Preuve: Par le théorème de Courcelles, il suffit de construire une formule monadique du second ordre qui caractérise ce problème. Voici une telle formule.

$diffusion(V, E, k, m, u) \equiv \exists A_1, \exists A_2, ..., \exists A_k (Correct(u, A_1) \wedge Correct(A_1, A_2) \wedge ... \wedge Correct(A_{k-1}, A_k) \wedge A_k = V))$. avec

$Correct(A, B) \equiv \exists C_1, \exists C_2, ..., \exists C_m, \forall u \in B \backslash A (ExisteValide(A, u, C_1, C_2, ...C_k) \wedge Disjoints(C_1, C_2) \wedge Disjoints(C_1, C_3) \wedge ... \wedge Disjoints(C_{k-1}, C_k)$. avec $Disjoints$ et $ExisteValide$:

$Disjoints(C, D) \equiv \forall u \in C, \forall v \in D, u \neq v$.

$ExisteValide \equiv Valide(A, u, C_1) \vee Valide(A, u, C_2) \vee ... \vee Valide(A, u, C_k))$. avec

$Valide(A, u, C) \equiv \exists v \in C, \exists w \in C, \forall x \in C, inc(v, A) \wedge inc(u, w) \wedge deg(v, C) = 1 \wedge deg(w, C) = 1 \wedge deg(x, C) = 2$.

□

Ce qui empêche de résoudre de déduire que le problème sans restriction sur le nombre m de chemins simultanés est linéaire, c'est le fait que dans la logique monadique de second ordre, il est impossible de faire $\exists k \in \Re$ et à la place, il faut tester tous les cas .Ici cela aboutit à une complexité exponentielle car le nombre de chemins possibles en une étape est proportionnel au nombre de sommets.

5 Diffusion dans les pathwidths 2

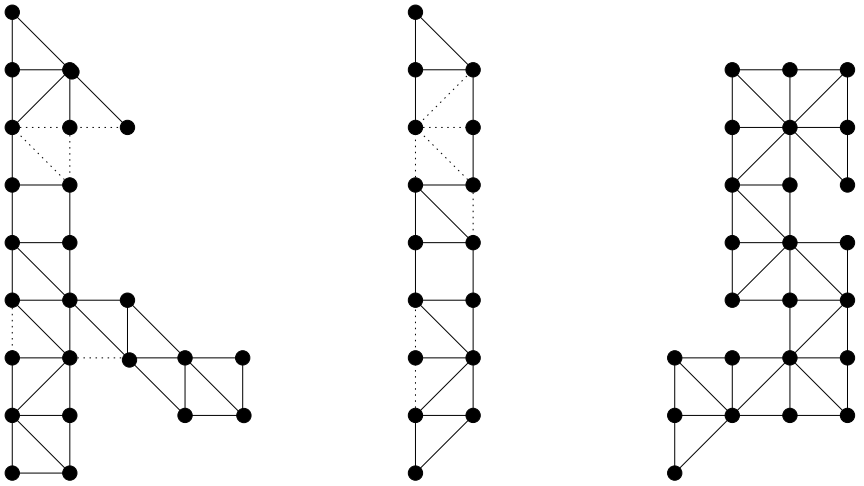
Après présentation des pathwidth 2, nous allons étudier le problème de diffusion sur ces graphes.

Définition 12 *Ce sont des treewidths 2 qui ont un graphe couvrant qui n'utilise pas l'opérateur $\oplus$. Les pathwidth 1 sont les chemins. Voici quelques exemples de treewidth 2 et de pathwidth 2:*

Voir les exemples de pathwidth 2 sur la page ci-contre.

Nous avons essayé de résoudre ce problème de diffusion dans les cas des treewidth 2. Par la suite, pour simplifier le problème, nous nous sommes contentés de raisonner sur les pathwidth 2. Raisonner sur les pathwidth nous permet de construire un algorithme qui rajoute les sommets un à un.

Nous avons essayé de concevoir un algorithme basé sur celui qui traite les arbres. Nous avons donc essayé de reconstruire une trace et redéfinir un ordre sur ces traces.

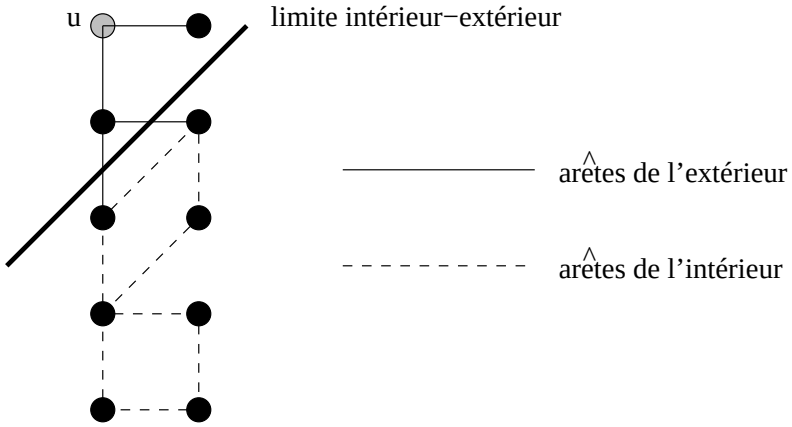


un graphe de treewidth 2 deux pathwidth 2.

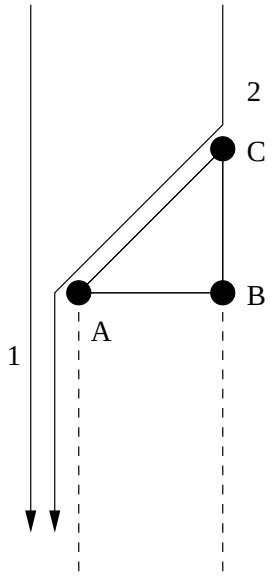
arêtes manquantes pour que ces graphes soient des 2−tree.

La première difficulté que nous avons rencontrée est que dans les pathwidth, plusieurs appels peuvent passer dans un sens. En effet, dans les arbres une seule arête relie le père au reste du graphe. Ici, plusieurs arêtes peuvent le faire donc le codage des appels n'est plus -1,0,1 mais ...,-3,-2,-1,0,1,2,3;....

Puis dans un graphe, pour définir l'intérieur, il suffisait de dire que c'était le sous-arbre qui avait comme racine le sommet à traiter. Ici dans les pathwidth, pour définir l'intérieur il faut deux sommets qui appartiennent à la même clique (et on considère que l'arête qui éventuellement les relie fait partie de l'intérieur).



Ensuite nous définissions une trace sur l'arête qui reliait le fils au père. Ici, nous devons toujours rajouter une arête mais nous pouvons rajouter dans certains cas deux arêtes à la fois donc la nouvelle trace que nous avons appelée empreinte prend en argument non plus un sommet mais deux. Également l'empreinte contient le codage des traces de deux sommets donc la fonction empreinte prend en indice le sommet dont on cherche la trace. Par exemple:

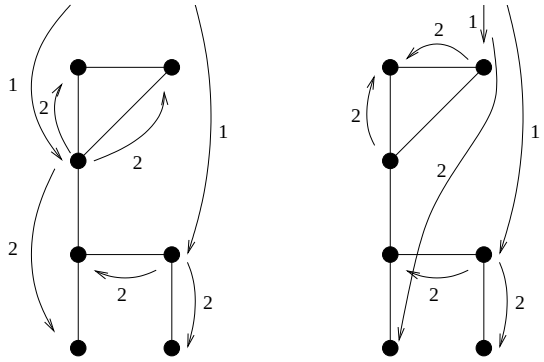


exemples dans le cas d'une diffusion en 2 étepes.

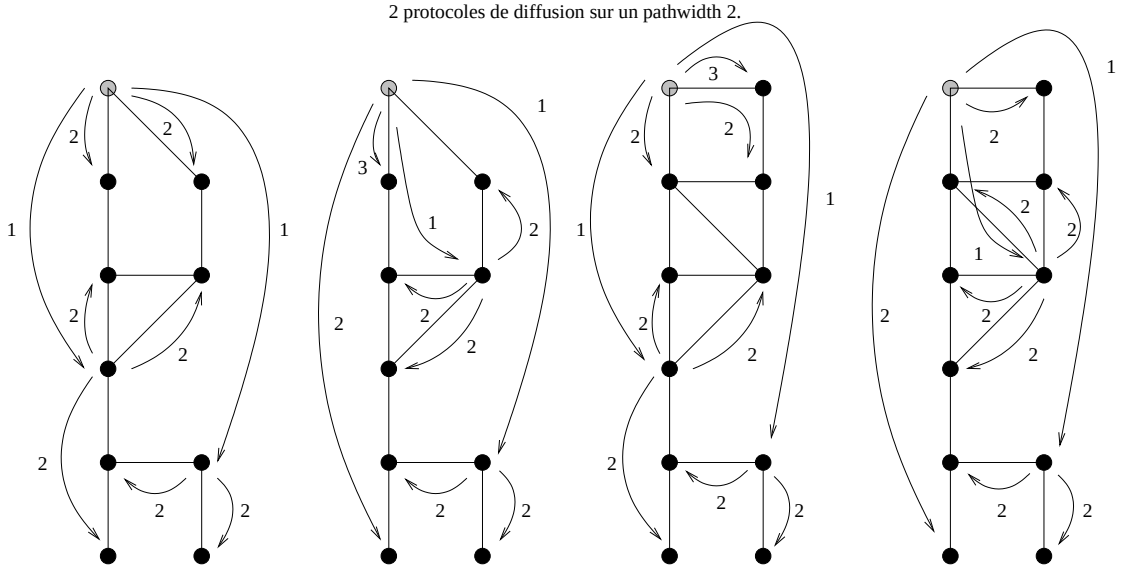
Dans cette exemple on a:
$empreinte_a$(a,b)=[1,1];
$empreinte_b$(a,b)=[0,0];
$empreinte_a$(a,c)=[1,0];
$empreinte_c$(a,c)=[0,1];
Cette empreinte permet également de reconstituer le protocole de diffusion. En effet, si a et b sont deux sommets d'un pathwidth 2 et c le sommet tel que a,b,c forment une clique dans le 2-tree dont est extrait le pathwidth 2 et tel que a fassent partie de l'intérieur de b,c, alors les communications de l'arête bc sont égales à $empreinte_b(b,a) - empreinte_b(b,c)$

Nous allons maintenant essayer de redéfinir l'ordre sur les protocoles, mais le problème que nous avons rencontré est donné par les deux protocoles ci-contre.

En effet, ces deux protocoles sont valides dans le sous-graphe et il existe deux graphes tel que dans l'un, un des protocoles est optimal et l'autre pas et pour le deuxième graphe c'est l'inverse. Ce qui veut dire que quand on raisonne au niveau du sous-graphe on est obligé de raisonner en conservant les deux protocoles. Autrement dit, il n'existe plus d'ordre total sur les protocoles



2 protocoles de diffusion sur un pathwidth 2.



comme dans le cas des arbres.

L'ordre que nous avons choisi sur les empreintes est le suivant:

Définition 13 *Soient L, L' deux protocoles sur un graphe G et a, b deux sommets de G*

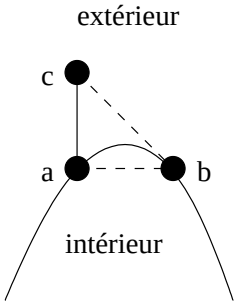
empreinte(L,a,b)≥ empreinte(L',a,b) si et seulement si
$empreinte_a(L,a,b) \geq empreinte_a(L',a,b)$ et
$empreinte_b(L,a,b) \geq empreinte_b(L',a,b)$

Cet ordre n'est pas total. Mais il vérifie la propriété suivante :

Soit L' un protocole, a et b deux sommets, soit V tel que:
V≥ $empreinte_a(L',a,b)$ alors ∃ L un protocole tel que $empreinte_a(L,a,b) = V$
et $empreinte_b(L,a,b) = empreinte_b(L',a,b)$

Algorithme 2 *Cet algorithme conserve une liste de protocoles et la met à jour en rajoutant les nouveaux sommets. Ici, on connaît $L_a = empreinte_a(b,a)$ et $L_b = empreinte_b(b,a)$ et on veut calculer $L'_b = empreinte_b(b,c)$ et $L'_c = empreinte_c(b,c)$. Nous considérons le cas par cas, selon la présence ou non des arêtes*

premier cas:

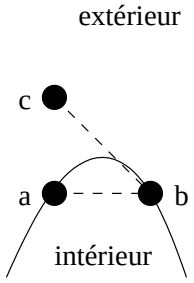


1)S'il n'existe pas d'arêtes entre c et b alors on pose $L'_c = L_a$ et $L'_b = L_b$

2)S'il y a une arête entre c et b alors il faut unir les appels entrants de a avec les appels sortants de b et vice-versa.

Maintenant, il faut choisir quand informer c, on chosi de l'informer en utilisant un appel de l'intérieur le plus tard possible. Dans le cas où cela ne l'est pas alors on l'informe en faisant un appel à l'extérieur le plus tard possible en c, mais dans le cas où l'on peut informer c par un chemin passant par b, il faut rajouter un nouveau protocole où on informe c par un appel entrant par b le plus tard possible. On crée donc au maximum, un nouveau protocole.

deuxième cas:



Si L_a fait un appel à l'extérieur alors ce protocole n'est plus valide, et on l'efface de la liste.

1) dans le cas où il n'y a pas d'arête entre b et c: alors $L'_c = [0, 0, ..., 1]$ et $L'_b = L_b$

2) dans le cas où il y a une arête entre b et c: alors il faut tester toutes les façons dont les appels entrants et sortants peuvent se répartir entre b et c: il faut tester toutes les combinaisons et après tester si on informe a ou b en premier.

6 Conclusion et perspectives

Dans ce rapport, nous avons donc introduit les notions de treewidth et de pathwidth.

Ensuite, nous avons énoncé les notions nécessaires à la compréhension du théorème de Courcelles (automates, automates d'arbre, t-boundaried graph, graphe couvrant, logique monadique du second ordre).

Après cela, nous avons pu énoncer le théorème de Courcelles.

Puis, nous nous sommes intéressés aux problèmes de diffusion et nous avons introduit l'algorithme dont nous nous sommes inspirés pour raisonner sur ce problème.

Par la suite, nous avons présenté les résultats que l'application du théorème de Courcelles nous a permi de trouver.

Enfin, nous avons conçu un algorithme pour résoudre le problème dans le cas général pour les pathwidth 2.

Mais, nous n'avons pas eu le temps de trouver une preuve que notre algorithme fonctionne effectivement en temps polynomial.

Également, il doit être possible d'étendre cet algorithme au treewidth en gérant en plus les noeuds entre les branches.

Le passage de treewidth 1 à treewidth 2 a fait perdre la propriété que l'ordre est total. Il faudrait donc voir comment se comporterait cet algorithme dans le cas de treewidth plus élevé.